JSD: Parallel Job Accounting on the IBM SP2

William Saphir[1], James Patton Jones[1]

NAS Technical Report NAS-95-16 July 95

NAS Scientific Computing Branch
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

Abstract

The IBM SP2 is one of the most promising parallel computers for scientific supercomputing — it is fast and usually reliable. One of its biggest problems is a lack of robust and comprehensive system software. IBM's software doesn't provide accounting for parallel jobs other than what is provided by AIX for the individual process components. Without parallel job accounting, it is not possible to monitor system use, measure the effectiveness of system administration strategies, or identify system bottlenecks. To address this problem, we have written **jsd**, a daemon that collects accounting data for parallel jobs. **jsd** records information in a format that is easily machine- and human-readable, allowing us to extract the most important accounting information with very little effort. **jsd** also notifies system administrators in certain cases of system failure.

1.0 Introduction

In July, 1994, a 160-node IBM SP2 was placed at the Numerical Aerodynamic Simulation (NAS) facility at NASA Ames Research Center. One of the research goals of the NAS SP2 testbed is to learn how to create a "production" parallel environment of the type provided on NAS' parallel vector Cray C90 supercomputers.

---

1. Computer Sciences Corporation, NASA Contract NAS 2-12961, Moffett Field, CA 94035-1000

The SP2 is essentially a collection of RS6000/590 workstations connected by a fast switch (which is not required, but which is part of the NAS system). IBM's Parallel Operating Environment (POE) software provides the "glue" to hold these workstations together. One of the key challenges for this software (and for any other software that creates a parallel computer from a network of single-processor nodes) is to recognize parallel applications as distinct entities, rather than as collections of individual Unix processes. Parallel applications should be started, scheduled, and terminated as a whole. Parallel operating system software that recognizes parallel applications in this way we term "parallel aware."

POE is moderately parallel aware. It allows parallel jobs to be started and killed as a unit and provides other facilities which we won't discuss here. All parallel jobs are registered with a Job Manager, which allocates collections of nodes for use by parallel applications. At NAS, POE is configured so that the job manager allocates a given node to only one parallel job, enforcing space-sharing and prohibiting time-sharing. There are many reasons for this policy. For instance, many parallel jobs run extremely inefficiently when timeshared due to load balancing and synchronization effects. Furthermore, only one process per node may access the switch in "user space" mode, which provides low latency and high bandwidth communication. Generally speaking, only embarrassingly parallel, dynamically load-balanced applications can perform efficiently when timeshared against each other. Of course individual processes in a parallel application are timeshared with Unix system processes on each node, but the impact of system processes is usually small.

Surprisingly, POE does not collect accounting data for parallel jobs. It is possible to collect standard Unix accounting information about individual processes, but this is not adequate. To understand system usage and adequately schedule resources, we need to know how many processes comprise a parallel application and what type of application it is (POE or PVMe, for example). There is no direct way to tell, for example, whether 16 Unix processes comprise a single parallel application or 2 applications of 8 processes each. Unix process information also has trouble capturing the basic intent of our allocation scheme, which is that a user owns a node for a given period of time, and that it doesn't matter what the user runs. For instance, if the user logs on to a node in order to attach a debugger, we are not interested in accounting for the separate debugger process. If PVM runs a daemon as well as a user program, we are actually interested in the wall clock time of the daemon, which determines how long the node is "owned," rather than the running time of any application programs. Sorting out what's interesting from what's not is not straightforward. Despite these problems, in principle we could probably obtain all the necessary information from AIX accounting data and process it carefully to get the interesting data. The method used in **jsd** is a lot simpler and a lot less work.

The simplest and most direct approach to collecting accounting information would be to have the Job Manager keep a log of all jobs. We hope that IBM will

eventually do this. Another approach would be to use accounting information provided by our batch scheduling system, PBS. While we will eventually use this information, PBS currently schedules only a portion of SP2 jobs and therefore doesn't provide a complete picture. Instead, we have written a simple monitoring daemon, **jsd**, which keeps a record of all parallel jobs run on the SP2.

2.0 JSD

jsd is a daemon that runs continuously and collects information about parallel jobs running on the SP2. It records information about individual jobs (owner, number of nodes, start and stop time, etc.) and "snapshot" information about the current system state. It also notifies system administrators when the Job Monitor fails to respond.

There is nothing fancy about the basic operation of **jsd**. It examines the state of the system at fixed time intervals (currently every 30 seconds), and updates its internal picture of what jobs are running. **jsd** notices when new jobs appear and when old ones disappear. Clearly **jsd** may not notice jobs lasting fewer than 30 seconds, and may overestimate the length of jobs by as much as 30 seconds. It will not underestimate because it uses a start time reported by the job manager.

jsd queries the Job Manager for its information, using an interface documented in `/usr/include/jm_client.h` and also used by the **jm_status** command. The routines in the jm_client interface allow a process to establish a connection to the Job Manager and to query the Job Manager. After some experimentation, we determined that the Job Manager forks a new server process to handle each connection. To avoid incurring this overhead for each request (every 30 seconds) **jsd** opens one connection and repeatedly queries the same connection. (Originally when opening a separate connection for each request, we exposed a bug in the jm_client software that caused connections to the SDR (System Data Repository) daemon to remain open and eventually crash the system.)

jsd creates three logs — a snapshot log, a job log, and an error log.

2.1 Snapshot Log

The snapshot log contains "snapshots" of system activity. A typical entry looks like this:

```
E  799444425     8   104       Tue May  2 12:53:45 1995
J      80102      user1      1       3749
J      58760      user2      4       8815
J      14041      user3      8       7922
J      19171      user4      4       2556
J      89546      user5     18       1689
J      17710      user6     32       1401
J      67061      user7      5        984
```

```
J     119664      user8     32        674
```

The format of the entry is designed for easy parsing by both humans and computers. The “E” in the first line denotes a new Entry in the log file. This line also records the number of jobs running and the total number of nodes in use. The date is recorded in standard Unix time (seconds since 12 A.M. Jan 1, 1970, GMT) and also in human-readable format. The Job Manager provides information that allows **jsd** to determine whether the job was started by poe, PVMe, or loadleveler. More specifically, it gives the name of the application that reserved the nodes through the Job Manager. Subsequent lines, starting with “J” denote job entries. Each line shows the process id of the controlling process, job owner, number of nodes, the length of time (in seconds) the job has been running.

2.2 Job Log

The job log contains a record of each job run on the system. A typical set of entries looks like this:

```
J  user1   16   799442310   1111  poe  Tue May  2 12:18:30 1995
J  user2   1    799442448   1064  poe  Tue May  2 12:20:48 1995
J  user3   1    799443488   86    poe  Tue May  2 12:38:08 1995
J  user2   31   799437126   6509  poe  Tue May  2 10:52:06 1995
J  user2   16   799443535   161   poe  Tue May  2 12:38:55 1995
```

Each line corresponds to a single job, and records the job owner, number of nodes, start time, total time (in seconds), and type of job.

The snapshot log and job log contain equivalent data, and one could generate one from the other. Why have both? The job log contains the essential accounting data, in a form that facilitates generation of reports based on user, number of nodes, or length of job. The snapshot log provides the same data in a way that facilitates generation of reports showing usage as a function of time, such as average number of nodes in use during each hour of the day. Furthermore, the snapshot log is updated continuously, so little important data is lost in the event of system crash, when a long-running job would not appear in the job log.

2.3 Job Manager failure detection

jsd notifies system administrators when its connection to the Job Manager is lost. Usually this is due to a Job Manager crash. It attempts to open a new connection every 30 seconds and notifies system administrators if it is successful. **jsd** often gives us our first indication of system failure.

At NAS, **jsd** utilizes the NAS Centralized Test Management System (CTMS) to report critical errors. CTMS is a tool that receives status and error messages from various systems and forwards all messages to a central point of access, which is

monitored 24 hours a day. **jsd** can also notify system administrators through electronic mail.

3.0 Report Generation

Both the job and snapshot logs are designed to be easily readable by both humans and computers. Our report generation utilities are short and straightforward shell scripts and C programs. The data generated by **jsd** has been extremely helpful in assessing system performance.

Figure 1 shows the utilization of the NAS SP2 over a three month period at the beginning of 1995. The utilization is expressed as a percentage of total node-hours available in a week. The total hours include system maintenance periods and unscheduled downtime, so that the utilization figures are conservative. The most interesting feature of Figure 1 is that it shows a large increase in utilization after the introduction of the PBS job scheduler.

Figure 2 shows hourly utilization over a one-week period. The significant drop-off in utilization around 5 A.M. was the result of a scheduling algorithm which tried to keep nodes “highly available” during prime time, which started at 6 A.M. PST. The schedule has since been modified to increase utilization in the early morning hours.

Figure 3 shows the node hours used by the top 50 users in March, 1995. The distribution is much more uniform than we had expected, and shows that many users are getting significant system usage.

Figure 4 gives the distribution of jobs by number of nodes over a 1 month period in 1995, showing millions of node-seconds used by jobs of different sizes. Note that large jobs use proportionately more node-seconds per unit of wall clock time. Figure 4 shows an expected peak at 32 nodes and smaller peaks at other

powers of two, but also shows very large peaks at 18, 20, 39, 100 and 144 nodes. Each of these peaks can be attributed to a particular application.

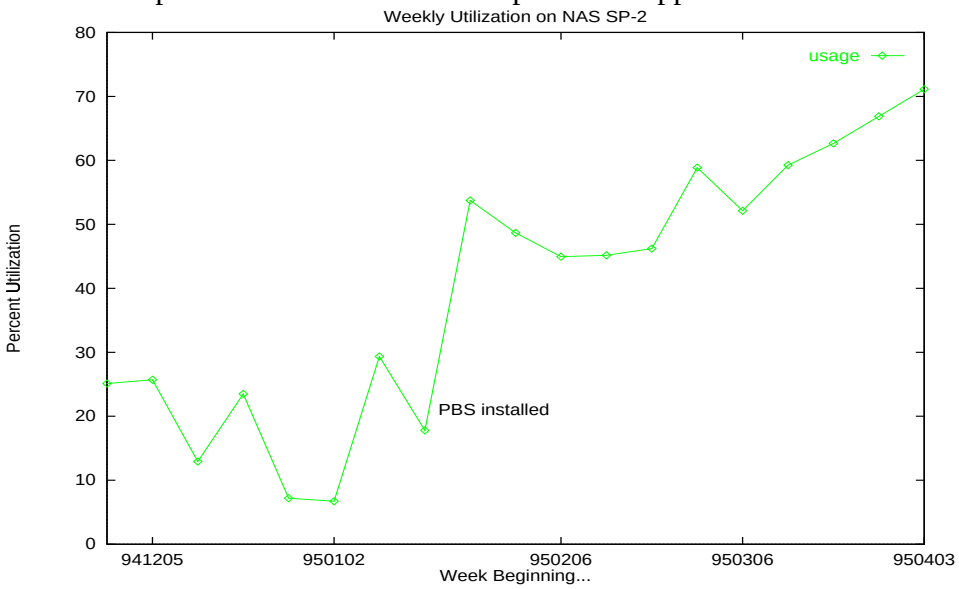

FIGURE 1.

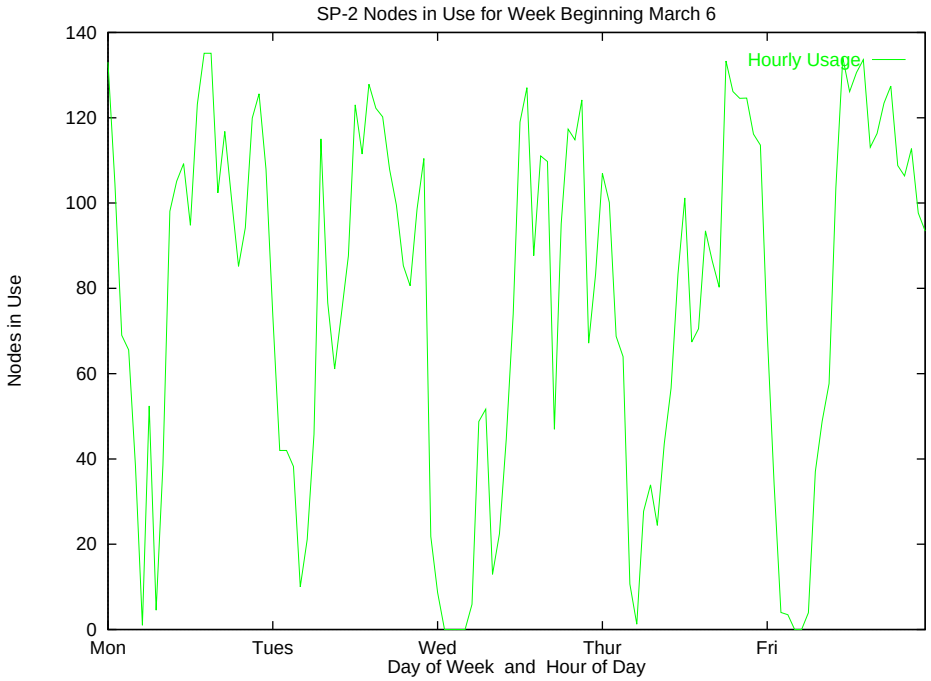

FIGURE 2.

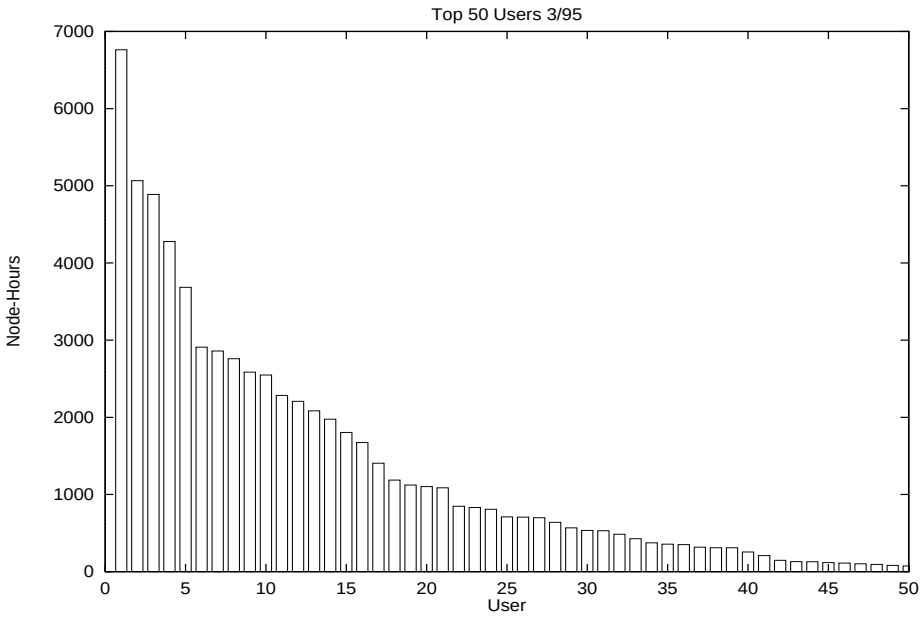

FIGURE 3.

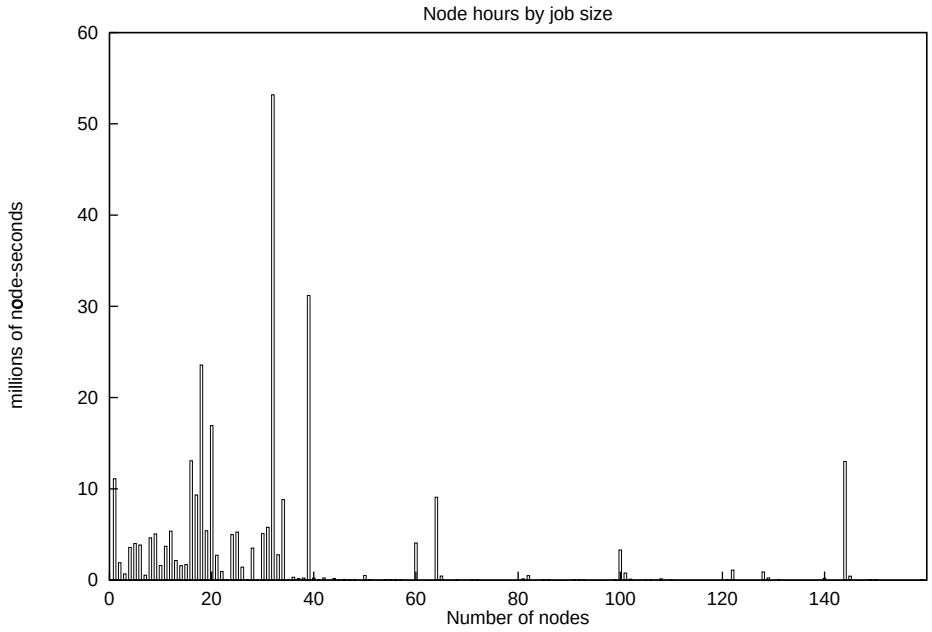

FIGURE 4.

4.0 Installation and Maintenance

jsd is simple to set up and run. Most sites will probably want to configure the following in the Makefile:

- Define CTMS_FLAG=-DCTMS if CTMS is being used
- Specify location of CTMS libraries if used (ignored if CTMS is not set)
- Define POC, a list of addresses of system administrators to whom to send mail
- Edit the locations of the log files. By default log files are written in /usr/adm/SPlogs (the names are set in the C source, not the Makefile)

By default, **jsd** is installed in `/usr/local/etc`. To have it automatically run at boot time and automatically restart on error, add the following line to /etc/inittab:

```
jsd:2:respawn:/usr/local/etc/jsd > /dev/console 2>&1
```

There is currently no protection against multiple **jsd** processes starting up at once. Multiple **jsd** processes will write to the same log files, with unpredictable results.

jsd should run on the control workstation or another front-end workstation to the SP2. It should write to a local filesystem, not one that is NFS-mounted.

5.0 Conclusions

jsd has proven to be a useful tool for parallel job accounting on the NAS SP2. Although it is only a temporary solution, while we await a more complete solution within POE, it provides most of the accounting information necessary to tune administrative policies and monitor usage.